IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Appellant:	Srinivasamurthy et al.	Patent Application	
Application No.:	10/630,913	Group Art Unit:	2192
Filed:	7/31/2003	Examiner:	Dao, Thuy Chan

For: Application Specific Optimization of Interpreters for Embedded Systems

APPEAL BRIEF

Table of Contents

I. Real Party in Interest

The assignee of the present invention is Hewlett-Packard Development Company, L.P.

II. Related Appeals and Interferences

There are no related appeals or interferences known to the Appellants.

III. Status of Claims

Claims 1, and 21-39 are pending, with claims 1 and 30 being independent. Claims 2-20 were previously cancelled. Claims 1 and 21-39 are rejected. This Appeal involves Claims 1 and 21-39.

IV. Status of Amendments

All proposed amendments have been entered. An amendment subsequent to the Final Action has not been filed.

V. Summary of Claimed Subject Matter

Independent Claims 1 and 30 of the present application pertain to embodiments associated with a method of and system for optimizing performance of an interpreter based runtime system. Reference to text (by page and line numbers) of the present application and figure elements (by reference number) of the present application that describe the claimed embodiments is provided below.

In Claim 1, "[a] method of optimizing performance of an interpreter based runtime system, the runtime system including a virtual machine, the virtual machine adapted to run an application in the context of the runtime environment" is recited. "[O]ptimizing the virtual machine based on semantics of the application to be run on the virtual machine, with at least a portion of the semantically enriched opcodes being specific to the application," as recited in Claim 1, is described at least by: Figure 2, blocks 21, 26, and 27; and Figure 6; page 5, lines 2-7; page 13, lines 27-28; and page 7, line 28 -page 8, line 1; semantically enriched opcode detection is described at least at page 9, line 1 - page 10, line 24; optimization is described at least at page 10, line 25 - page 12, line 3. "[A]ugmenting a bytecode set of the virtual machine with semantically enriched opcodes, thereby constituting an application domain-specific virtual machine," as recited in Claim 1, is described at least by: page 3, lines 5-7; page 4, lines 8-11; page 7, line 28 - page 8, line 1; and Figure 2, target machine 24 and blocks 21, 26, and 27. "[P]erforming a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off," as recited in Claim 1, is described at least by: page 3, lines 21-23; page 13, lines 17-21; and claim 9 as originally filed. [A]analyzing frequently executed bytecodes and encoding the

semantically enriched opcodes into interpreter action codes of the instruction set of the virtual machine to efficiently decode the frequently executed bytecodes," as recited in Claim 1, is described at least by: Claim 15 as originally filed; page 7, line 28 - page 8, line 6; page 9, lines 21-30; and Figure 4B. [O]ptimizing a translation by the interpreter action codes of the semantically enriched opcodes according to a system state, said system state being represented by at least one symbolic variable," as recited in Claim 1, is described at least by: page 11, line - page 12, line 3; Figure 2 (sEc code generation); and Figure 6. [S]tatically embedding the semantically enriched opcode to optimize execution of the interpreter-based runtime system," as recited in Claim 1, is described at least by: Figure 2, sEc embedding 22; page 3, lines 15-17; page 8, lines 7-10; and page 12, lines 5-21.

In Claim 30, "[a] system for optimizing performance of an interpreter based runtime system, the runtime system including a virtual machine" is recited. "[A]n application, as recited in Claim 30, is described at least by: page 5, lines 14-19; page 6, lines 8-9; Figure 1, Java Application 10; and Figure 2, JVM source 27. "[A]n embedded processor," as recited in Claim 30, is described at least by: page 1, lines 14-21; page 5, lines 14-15; page 5, lines 26-28; and page 6, lines 26-30. "[A] a virtual machine configured to translate said application code into native machine code compatible with said embedded processor," as recited in Claim 30, is described at least by: page 10, line 25 - page 11, line 1; Figure 1, JVM 11; and page 6, lines 4-11. "[A] detection module, said detection module being configured to analyze said application code to identify code segments that could be efficiently represented as semantically enriched opcodes, at least a portion of said semantically enriched opcodes being specific to said application: Figure 2, sEc detection 20; and page 9, line 1 - page 10, line 24. [A]n embedding module, said embedding module being configured to embed said

semantically enriched opcodes in said application," as recited in Claim 30, is described at least by: Figure 2, embedding 22; page 3, lines 15-17; page 8, lines 7-10; page 12, lines 5-31; and page 13, line 29 - page 14, line 10. [A] code generation module, said code generation module being configured to generate optimized action code for translating said semantically enriched opcodes according to symbolic states, each of said symbolic states being represented by at least one symbolic variable," as recited in Claim 30, is described at least by: page 8, lines 7-9; page 10, line 25 - page 12, line 3; Figure 2 (sEc code generation 21); and Figure 6. [A] build module configured create an application domain-specific virtual machine by incorporating said optimized action code and a bytecode set comprising said semantically enriched opcodes into said virtual machine," as recited in Claim 30, is described at least by: Figure 2 (build); and page 12, lines 5-31.

VI. Grounds of Rejection to Be Reviewed on Appeal

1. Whether Claims 1 and 21-39 are unpatentable under 35 U.S.C. §103(a) over U.S. Patent 6,988,261 to Sokolov et al. (hereinafter "Sokolov") in view of U.S. Patent 6,014,519 to Egashira.

VII. Argument

1. Whether Claims 1 and 21-39 are unpatentable under 35 U.S.C. §103(a) over Sokolov in view of Egashira.

Appellants have reviewed the cited art and submit that Claims 1 and 30 are not rendered unpatentable in view of the cited art for at least the following rationale.

Obviousness Requirements

"As reiterated by the Supreme Court in *KSR*, the framework for the objective analysis for determining obviousness under 35 U.S.C. 103 is stated in *Graham v. John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966). Obviousness is a question of law based on underlying factual inquiries" including "[a]scertaining the differences between the claimed invention and the prior art" (MPEP 2141(II)). "In determining the differences between the prior art and the claims, the question under 35 U.S.C. 103 is not whether the differences themselves would have been obvious, but whether the claimed invention as a whole would have been obvious" (emphasis in original; MPEP 2141.02(I)). Applicants note that "[t]he prior art reference (or references when combined) need not teach or suggest all the claim limitations, however, Office personnel must explain why the difference(s) between the prior art and the claimed invention would have been obvious to one of ordinary skill in the art" (emphasis added; MPEP 2141(III)).

Claim 1

Attention is directed to Independent claim 1 which recites, emphasis added:

> A method of optimizing the performance of an interpreter based runtime system, the runtime system including a virtual machine, the virtual machine adapted to run an application in the context of the runtime environment, the method comprising:

augmenting a bytecode set of the virtual machine with semantically enriched opcodes, thereby constituting an application domain-specific virtual machine;

optimizing the virtual machine based on semantics of the application to be run on the virtual machine, with at least a portion of the semantically enriched opcodes being specific to the application;

performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off;

analyzing frequently executed bytecodes and encoding the semantically enriched opcodes into interpreter action codes of the instruction set of the virtual machine to efficiently decode the frequently executed bytecodes;

optimizing a translation by the interpreter action codes of the semantically enriched opcodes into a native code according to a system state, said system state being represented by at least one symbolic variable; and

statically embedding the semantically enriched opcode to optimize execution of the interpreter-based runtime system.

The Office Action (page 12) concedes that Sokolov does not disclose "performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off," (emphasis added) as is recited in Claim 1 Consequently, the Office Action cites to Egashira, and in particular to Figure 3; col. 7, line 60 - col. 8, line 53; Figure 7; col. 12, line 57 - col. 13, line 18, and col. 6, line 60 - col. 7, line 33 of Egashira for teachings to support obviousness of this feature. Appellants agree that Sokolov does not disclose this feature. However, Appellants submit that Egashira fails to remedy the shortcomings of Sokolov. Specifically, Appellants submit that neither Egashira nor Sokolov in view of Egashira teaches suggests performing any sort of quantitative trade-off between execution time and memory space.

While Appellants understand that Egashira may discuss pre-designating a code size that a programmer desires (see 201 of Figure 3 and col. 6, line 60 - col. 7, line 24) and having a priority on execution time (206 of Figure 3 and col.7, lines 25-40), Egashira appears to be

silent with respect to performing any sort of trade-off between the two to determine effective semantically enriched opcodes or to perform a "quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes," as is recited in Claim 1. Without any discussion (and there is none, only citations), Appellants submit that the rejection of this feature of Claim 1 appears to be based, at best, on conclusory assumptions. Per MPEP 2141(III), "[R]ejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness," emphasis added. In re Kahn, 441 F. 3d 977, 988 as cited by KSR International Co. v. Teleflex Inc. (KSR), 550 U.S. 398, 82 USPQ2d at 1396 (2007). For at least this reason, Appellants submit that the combination of Sokolov in view of Egashira fails to teach, suggest, or otherwise render obvious Appellants' Claim 1.

Claims 1 and 30

Furthermore Appellants submit that Sokolov fails to teach or suggest "statically embedding the semantically enriched opcode to optimize execution of the interpreter-based runtime system," as recited in Claim 1 and similarly in Claim 30 (embedding module). At page 8, the "Office Action" (10/22/2009) asserts that Sokolov discloses this feature at Figure 4, block 408; col. 6, line 66 - col. 7, line 19; Figure 5; and col. 7, lines 37-67. Appellants disagree that this or any portion of Sokolov teaches the above recited feature. Instead, Appellants submit that the cited portion (see at least col. 7, lines 20-67) appears to teach away from the above recited feature of "statically embedding the semantically enriched opcode..." by disclosing a dynamic process and principle of operation where,

> ... a sequence of two or more Java TM Bytecode instructions 506 in the stream 504 can be identified by the Java™ Bytecode verifier 500 ... the

> Java™ Bytecode verifier can operate to replace the sequence with a Java™ macro instructions 508 (I1-IM)... [a]s a result, the stream 504 is reduced to a stream 510 consisting of (N-M) Java™ Bytecode instructions.

Appellants submit that this process described in Sokolov would need to be dynamically repeated each type an application is run, which appears to Appellants to be the opposite of "statically embedding the semantically enriched opcode to optimize execution of the interpreter-based runtime system," as is recited in Claim 1 and similarly in Claim 30 (see also page 12, lines 17-21 of Appellants' specification as filed). Thus, Appellants submit that Sokolov does not teach or disclose the above recited feature of Appellants' Claim 1 or Claim 30.

Moreover, according to MPEP 2143.01, "[i]f the proposed modification or combination of the prior art would change the principle of operation of the invention being modified, then the teachings of the references are not sufficient to render the claims *prima facie* obvious. *In re Ratti*, 270 F.2d 810, 123 USPQ 349 (CCPA 1959)" (emphasis added). Per Appellants' understanding, the Office Action's proposed interpretation/modification of Sokolov from a dynamic system to a static system (as described in Appellants' Claim 1 and Claim 30) would appear to change the principle of operation of Sokolov from a system which operates on streams of data that are being processed to a system that statically embeds macro instructions into applications. Appellants submit that such a change to the principle of operation of Sokolov is impermissible in an effort to make a *prima facie* case of obviousness.

Appellants submit that neither Egashira, nor Sokolov in view of Egashira cures this deficiency of Sokolov. Firstly, Appellants submit that Egashira was not relied upon to teach the feature of "statically embedding the semantically enriched opcode to optimize execution

of the interpreter-based runtime system," as recited in Claim 1 and similarly in Claim 30. Secondly, Appellants submit that even if Egashira were to disclose such a feature, it would be not be permissible to modify Sokolov in view of Egashira to arrive at the Appellants' feature of "statically embedding the semantically enriched opcode to optimize execution of the interpreter-based runtime system," as, has previously described, such modification would change the principle of operation of Sokolov.

As such, Applicants submit that Sokolov in view of Egashira fails to make a *prima facie* case of obviousness as not all of the features of Claims 1 or 30 are taught or suggested by the combination and as the cited combination appears to teach away from the Office Action's proposed modification. Additionally, and as required by the MPEP as cited above, the Office Action fails to explain why the identified differences between Applicants' claimed invention and Sokolov in view of Egashira would have been obvious to one of ordinary skill in the art.

In view of the combination of Sokolov in view of Egashira not satisfying the requirements of a *prima facie* case of obviousness, Appellants respectfully submit that independent Claims 1 and 30 overcome the rejections under 35 U.S.C. §103(a), and that these claims are thus in a condition for allowance. Appellants respectfully submit the combination of Sokolov in view of Egashira also does not teach or suggest the claimed embodiments as recited in Claims 21-29 that depend from independent Claim 1 or Claims 31-39 that depend from independent Claim 30. Therefore, Appellants respectfully submit that Claims 21-29 and 31-39 overcome the rejection under 35 U.S.C. §103(a) to Sokolov in view of Egashira,

and are in a condition for allowance by virtue of their dependence from allowable base claims.

Conclusion

The Appellants believe that pending Claims 1 and 21-39 are patentable over the cited art. Appellants respectfully request that the Board reverse the rejection of Claims 1 and 21-39.

The Appellants wish to encourage the Examiner or a member of the Board of Patent Appeals to telephone the Appellants' undersigned representative if it is felt that a telephone conference could expedite prosecution.

Respectfully submitted,
WAGNER BLECHER LLP

Dated: 02/18/2010 /John P. Wagner, Jr./

John P. Wagner, Jr.
Registration No.: 35,398

WAGNER BLECHER LLP
Westridge Business Park
123 Westridge Drive
Watsonville, CA 95076

Phone: (408) 377-0500

VIII. Appendix - Clean Copy of Claims on Appeal

1. A method of optimizing the performance of an interpreter based runtime system, the runtime system including a virtual machine, the virtual machine adapted to run an application in the context of the runtime environment, the method comprising:

augmenting a bytecode set of the virtual machine with semantically enriched opcodes, thereby constituting an application domain-specific virtual machine;

optimizing the virtual machine based on semantics of the application to be run on the virtual machine, with at least a portion of the semantically enriched opcodes being specific to the application;

performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off;

analyzing frequently executed bytecodes and encoding the semantically enriched opcodes into interpreter action codes of the instruction set of the virtual machine to efficiently decode the frequently executed bytecodes;

optimizing a translation by the interpreter action codes of the semantically enriched opcodes according to a system state, said system state being represented by at least one symbolic variable; and

statically embedding the semantically enriched opcode to optimize execution of the interpreter-based runtime system.

21. The method of claim 1, further comprising analyzing an application code using static optimization, the static optimization comprising parsing the application code to

identify at least one repeated sequence of bytecodes and replace the at least one repeated sequence of bytecodes with a semantically enriched opcode.

22. The method of claim 1, further comprising analyzing an application code using dynamic optimization, the dynamic optimization comprising:

analyzing temporal behavior of the application code during execution; and

identifying and replacing at least one repeated computational sequence in a bytecode stream with a semantically enriched opcode.

23. The method of claim 1, further comprising:

discovering at least one repetitive computational sequence used in a symbolic state; and

generating a semantically enriched opcode corresponding to the at least one repetitive computational sequence used in the symbolic state.

24. The method of claim 23, wherein the symbolic state comprises at least one of: control flow, data flow, entry points, and operational arguments.

25. The method of claim 1, further comprising optimizing native code output by the virtual machine using a global optimizing compiler.

26. The method of claim 1, further comprising optimizing the virtual machine by offline compilation of the virtual machine by a host device.

27. The method of claim 1, further comprising optimizing the virtual machine by online modification of a generic virtual machine by inserting a stub, the stub automatically loading a semantically enriched opcode when the virtual machine encounters an identified code segment with a bytestream.

28. The method of claim 1, further comprising offline embedding of the semantically enriched opcodes in an application by substituting a semantically enriched opcode for a corresponding code segment.

29. The method of claim 1, further comprising online embedding of semantically enriched opcodes by a class loader, said class loader substituting a semantically enriched opcode for a corresponding code segment.

30. A system for optimizing performance of an interpreter based runtime system, the runtime system including a virtual machine, the system comprising:

application code;

an embedded processor;

a virtual machine configured to translate said application code into native machine code compatible with said embedded processor;

a detection module, said detection module being configured to analyze said application code to identify code segments that could be efficiently represented as semantically enriched opcodes, at least a portion of said semantically enriched opcodes being specific to said application;

an embedding module, said embedding module being configured to embed said semantically enriched opcodes in said application;

a code generation module, said code generation module being configured to generate optimized action code for translating said semantically enriched opcodes according to symbolic states, each of said symbolic states being represented by at least one symbolic variable; and

a build module configured create an application domain-specific virtual machine by incorporating said optimized action code and a bytecode set comprising said semantically enriched opcodes into said virtual machine.

31. The system of claim 30, wherein said detection module is configured to analyze said application code using static optimization, said static optimization comprising parsing said application code to identify at least one repeated sequence of bytecodes and replace said at least one repeated sequence of bytecodes with a semantically enriched opcode.

32. The system of claim 31, wherein said detection module is further configured to analyze said application code using dynamic optimization, said dynamic optimization comprising:

analyzing temporal behavior of the application code during execution; and

identifying and replacing at least one repeated computational sequence in a bytecode stream with a semantically enriched opcode.

33. The system of claim 30, wherein said generation module is further configured to:

discover at least one repetitive computational sequence used in a said symbolic state; and

generate a semantically enriched opcode corresponding to the at least one repetitive computational sequence used within said symbolic state.

34. The system of claim 33, wherein said symbolic state comprises at least one of: control flow, data flow, entry points, and operational arguments.

35. The system of claim 30, further comprising a global optimizer compiler configured to optimizing native code output by said virtual machine.

36. The system of claim 30, wherein said virtual machine is compiled offline by a host device.

37. The system of claim 30, wherein said virtual machine is modified online by inserting a stub, said stub automatically loading a semantically enriched opcode when said virtual machine encounters an identified code segment with a bytestream.

38. The system of claim 30, wherein said application code is modified offline by substituting a semantically enriched opcode for a corresponding code segment contained with said application code.

39. The system of claim 30, wherein a class loader embeds said semantically enriched opcodes online, said class loader substituting a semantically enriched opcodes for a corresponding code segment.

IX. Evidence Appendix

No evidence is herein appended.

X. Related Proceedings Appendix

No related proceedings.